CHAPTER 5

UNIT V

Decision Trees and Ensemble Learning

Syllabus

Decision Trees : Impurity measures, Feature Importance. Decision Tree Classification with Scikit-learn, Ensemble Learning-Random Forest, AdaBoost, Gradient Tree Boosting, Voting Classifier.

Clustering Fundamentals : Basics, K-means: Finding optimal number of clusters, DBSCAN, Spectral Clustering. Evaluation methods based on Ground Truth- Homogeneity, Completeness, Adjusted Rand Index.

Introduction to Meta Classifier : Concepts of Weak and eager learner, Ensemble methods, Bagging, Boosting, Random Forests.

5.1 Decision Trees

Suppose a database D is given as $D = \{ t_1, t_2, \ldots, t_n \}$ and a set of desired classes are $C = \{ C_1, \ldots, C_m \}$, the **Classification problem** is to define the mapping m in such a way that which tuple of database D belongs to which class of C. Actually divides D into **equivalence classes**.

Classification Example

How teacher gives grades to students based on their marks obtained:

If $x >= 90$ then grade = A.

If $80 <= x < 90$ then grade = B.

If $70 <= x < 80$ then grade = C.

If $60 <= x < 70$ then grade = D.

If $x < 60$ then grade = E.

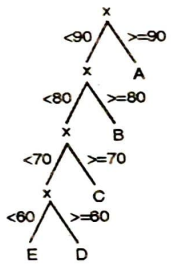


Fig. 5.1.1 : Classification of grading

- Decision tree learning is appropriate for problems having characteristics given below :
- Instances are represented by a fixed set of attributes (e.g. Gender) and their values (e.g. male, female) described as **attribute-value pairs**.
- If the attribute has small number of disjoint possible values (e.g. high, medium, low) or there are only two possible classes (e.g. true, false) then decision tree learning is easy.
- Extension to decision tree algorithm also handles real valued attributes (e.g. salary).
- Decision tree gives a class label to each instance of the dataset.

- If some of the examples have unknown values, still decision tree method can be used (Example: Wind value is known for few tuples).
- Decision trees or if-then rules represent learned functions which improve readability.

Decision Tree Representation

- Decision tree has leaf nodes and decision nodes.
- Every branch's last node is a leaf node which represents the class label.
- Decision node is used to take decision for the class label after some trials. It has a leaf node or sub tree.
- Play Tennis example for representing decision Trees.

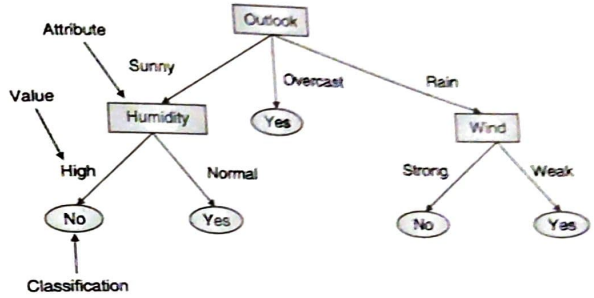


Fig. 5.1.2 : Representation of decision tree

Binary Decision tree

- It is a structure based on a sequential decision process.
- It goes from the root and based on evaluating features one of the two branches is chosen.
- Repeat the same procedure until a final leaf node or classification target is reached.

5.1.1 Impurity measures

1. Gini impurity index

- Suppose all attributes are continuous-valued.
- Assume that each value of an attribute has many possible splits.
- It can be adapted for categorical attributes.
- Gini is used in CART (Classification and Regression Trees), IBM's IntelligentMiner system, SPRINT (Scalable PaRallelizable INduction of decision Trees).
- If a data set T contains examples from n classes, Gini index, Gini (T) is defined as

$$\text{gini}(T) = 1 - \sum_{j=1}^{n} p_j^2$$

Where, p_j is the relative frequency of class j in T.

- gini(T) is minimized if the classes in T are skewed.

- Gini index of split is defined below after split T1 and T2 of sizes N1 and N2 respectively.

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- For every attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the attribute providing smallest $gini_{split}$ (T) is picked out to split the node. For continuous-valued properties, each possible split-point must be believed.
- The Gini impurity measures the probability of a misclassification if a label is randomly selected using the probability distribution of the branch. If all the samples belong to a single category, then index reaches its minimum (0.0).

2 Cross-entropy impurity index

- **Cross-entropy impurity index** is based on Information theory.
- When all the samples belong to one class are represented in a split, then null values are assumed. But if there is uniform distribution, then it is maximum.
- It permits to select the split which actually minimizes uncertainty about classification.

$$\text{Entropy} = -\sum_j p_j \log_2 p_j$$

- Information Gain (IG), determine **which attribute** in a given set of training feature vectors is **most useful**. We will use it to decide the ranking of attributes in the nodes of a decision tree.
- The **Information Gain (IG)** can be defined as follows:
- All attributes are considered to be categorical.
- It can be adapted for continuous-valued attributes.
- The attribute which has the highest information gain is selected for split.
- Assume there are two classes, P and N.
- Consider S samples, out of these p samples belongs to class P and n samples belongs to class N.
- The amount of information, needed to decide if random example in S belongs to P or N is defined as

$$I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

- Assume that using attribute A, a set *S* will be partitioned into sets $\{S_1, S_2, ..., S_v\}$.
- If S_i contains p_i examples of *P* and n_i examples of *N*, the entropy, or the expected information needed to classify objects in all sub trees S_i is

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

Entropy (E)

- The expected amount of information (in bits) needed to assign a class to a randomly drawn object in S under the optimal, shortest-length code.
- **Calculate information gain i.e. gain (A) :** Measures reduction in entropy, achieved because of the split. Take the split that achieves most reductions (maximizes GAIN)

$$\text{Gain}(A) = I(p, n) - E(A)$$

3. Misclassification impurity index

- The misclassification impurity is the simplest index, defined as :

$$I_E = 1 - \max\{p(i|t)\},$$

- It is a useful criterion for pruning but not recommended for growing a decision tree since it is less sensitive to changes in the class probabilities of the nodes.

5.1.2 Feature Importance

- To predict the output value, feature importance helps to evaluate significance of every feature of multi-dimensional dataset.
- Decision trees offer a different approach based on the impurity reduction determined by every single feature.
- In particular, considering a feature x_i, its importance can be determined as :

$$\text{Importance}(x_i) = \sum_k \frac{N_k}{N} \Delta I_{x_i}$$

- The sum is extended to all nodes where x_i is used, and N_k is the number of samples reaching the node k.

Sample code for feature importance in python

Considering the iris example and converting to a pandas.DataFrame()

```
from sklearn. tree import DecisionTreeClassifier
from sklearn. datasets import load_iris
import pandas as pd

clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
iris_pd = pd.DataFrame(iris.data, columns=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
clf = clf.fit(iris_pd, iris.target)
print(dict(zip(iris_pd.columns, clf.feature_importances_)))

Output : Showing the feature importance of each feature
{'sepal_length': 0.0, 'sepal_width': 0.013333333333333329, 'petal_length': 0.06405595813204505, 'petal_width':
0.9226107085346216}
```

- Gini index of split is defined below after split T1 and T2 of sizes N1 and N2 respectively.

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- For every attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the attribute providing smallest $gini_{split}(T)$ is picked out to split the node. For continuous-valued properties, each possible split-point must be believed.
- The Gini impurity measures the probability of a misclassification if a label is randomly selected using the probability distribution of the branch. If all the samples belong to a single category, then index reaches its minimum (0.0).

2 Cross-entropy impurity index

- **Cross-entropy impurity index** is based on Information theory.
- When all the samples belong to one class are represented in a split, then null values are assumed. But if there is uniform distribution, then it is maximum.
- It permits to select the split which actually minimizes uncertainty about classification.

$$\text{Entropy} = -\sum_j p_j \log_2 p_j$$

- Information Gain (IG), determine **which attribute** in a given set of training feature vectors is **most useful**. We will use it to decide the ranking of attributes in the nodes of a decision tree.
- The **Information Gain (IG)** can be defined as follows:
- All attributes are considered to be categorical.
- It can be adapted for continuous-valued attributes.
- The attribute which has the highest information gain is selected for split.
- Assume there are two classes, P and N.
- Consider S samples, out of these p samples belongs to class P and n samples belongs to class N.
- The amount of information, needed to decide if random example in S belongs to P or N is defined as

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- Assume that using attribute A, a set *S* will be partitioned into sets $\{S_1, S_2, ..., S_v\}$.
- If S_i contains p_i examples of *P* and n_i examples of *N*, the entropy, or the expected information needed to classify objects in all sub trees S_i is

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

Entropy (E)

- The expected amount of information (in bits) needed to assign a class to a randomly drawn object in S under the optimal, shortest-length code.
- **Calculate Information gain i.e. gain (A)** : Measures reduction in entropy, achieved because of the split. Take the split that achieves most reductions (maximizes GAIN)

$$\text{Gain}(A) = I(p, n) - E(A)$$

3. Misclassification impurity index

- The misclassification impurity is the simplest index, defined as :

$$I_E = 1 - \max\{p(i|t)\},$$

- It is a useful criterion for pruning but not recommended for growing a decision tree since it is less sensitive to changes in the class probabilities of the nodes.

5.1.2 Feature Importance

- To predict the output value, feature importance helps to evaluate significance of every feature of multi-dimensional dataset.
- Decision trees offer a different approach based on the impurity reduction determined by every single feature.
- In particular, considering a feature x_i, its importance can be determined as :

$$\text{Importance}(x_i) = \sum_k \frac{N_k}{N} \Delta I_{x_i}$$

- The sum is extended to all nodes where x_i is used, and N_k is the number of samples reaching the node k.

Sample code for feature importance in python

Considering the iris example and converting to a pandas.DataFrame()

```
from sklearn. tree import DecisionTreeClassifier
from sklearn. datasets import load_iris
import pandas as pd

clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
iris_pd = pd.DataFrame(iris.data, columns=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
clf = clf.fit(iris_pd, iris.target)
print(dict(zip(iris_pd.columns, clf.feature_importances_)))
```

Output : Showing the feature importance of each feature

```
{'sepal_length': 0.0, 'sepal_width': 0.013333333333333329, 'petal_length': 0.06405595813204505, 'petal_width':
0.9226107085346216}
```

5.1.3 Decision Tree Classification with Scikit-learn

Consider iris dataset , using the DecisionTreeClassifier class of Scikit-learn for binary decision tree.

```
from sklearn.datasets import load_iris
from sklearnimporttree
iris=load_iris()
clf=tree.DecisionTreeClassifier()
clf=clf.fit(iris.data,iris.target)

# export the tree in Graphviz format using the export_graphviz
import graphviz
dot_data=tree.export_graphviz(clf,out_file=None)
graph=graphviz.Source(dot_data)
graph.render("iris")
dot_data=tree.export_graphviz(clf, out_file=None, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True,rounded=True,. special_characters=True)
graph=graphviz.Source(dot_data)
graph
```

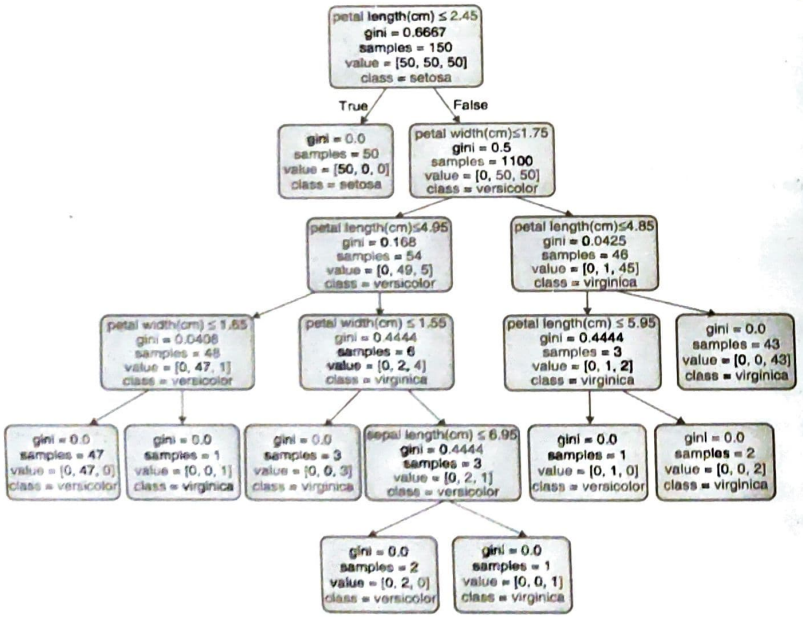


Fig. 5.1.3 : Graph of iris example

5.1.4 Ensemble Learning

Ensemble learning combines various set of learners (individual models) together which actually improvise on the stability and predictive power of the model. Combining classifier is an ensemble method which increases the accuracy. To get new improved model M*, combine a series of n learned models, M_1 , M_2 , ..., M_n. Popular ensemble methods are discussed as follows :

1. **Bagged (or Bootstrap) trees**

- Base learners are generated in parallel, so it is a parallel ensemble method (e.g. Random Forest).
- The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by averaging.
- For example, we can train M different trees on different subsets of the data (chosen randomly with replacement) and compute the ensemble :

$$f(x) = 1/M \sum_{m=1}^{M} f_m(x)$$

- In generalized bagging, different learners can be used in different population to reduce the variance error.

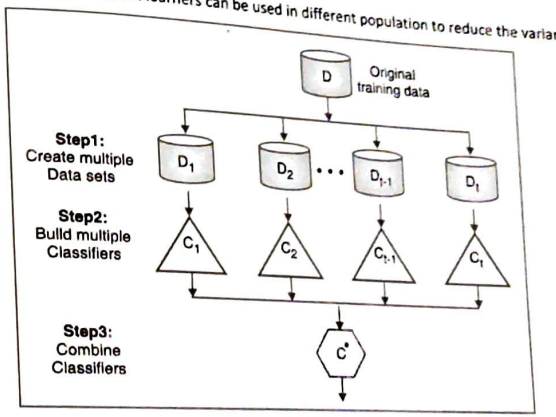


Fig. 5.1.4 : Bagging or bootstrap

- It averages the prediction from the accumulation of various classifiers used.
- A bootstrap method is used, for data set D of n tuples, for each iteration n tuple are sampled with replacement from D.
- In every iteration , a classifier model M is learned from training data set
- For unknown sample Y, each classifier gives class prediction.
- The bagged classifier M* uses the voting method i.e. the sample tuple Y is assigned the class with the most votes to tuple Y.
- For continuous values, it can be used for prediction by taking the average of all predictions for a given sample.

2. Boosted trees

- It is a family of algorithms that are able to convert weak learners to strong learners.
- It is a sequential ensemble method where the base learners are generated sequentially (e.g. AdaBoost).
- The basic motivation of sequential methods is to **exploit the dependence between the base learners.** The overall performance can be boosted by weighing previously mislabelled examples with higher weight.
- In boosting, each training tuple has weight.
- n number of classifiers are learned iteratively.
- After learning of M_i classifier, every time the weights are updated for next classifier learning i.e. M_{i+1}. So if the tuples which were misclassified by M_i will get higher weight for next classifier.
- Use voting method, where check the votes of each classifier to get the final M* which helps to get the accuracy.
- The extended boosting algorithm works for the prediction of continuous values.
- Boosting tends to accomplish greater accuracy as compared to bagging, there is a risk of overfitting the model.

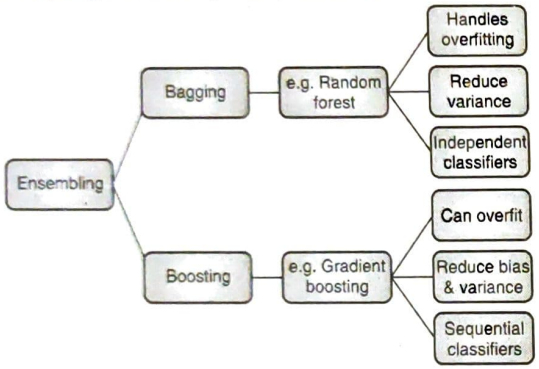


Fig. 5.1.5 : Ensemble Learning

5.1.4(A) Random Forest

- It is a supervised classification algorithm.
- It creates the forest with a number of trees as the name suggests.
- If **the number** of trees in the forest are more, the **accuracy** result is high.
- Random forest handles the missing values.
- Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features.
- The algorithm selects random subset of features to split the node.

Random Forest pseudocode

1. Select **"k"** features randomly from total **"m"** features where **k << m**.
2. Using the bet split point, calculate the node d from selected k features.
3. Using the best split, split the node d into child nodes.

4. Repeat **1 to 3** steps until "l" number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for "n" number times to create **"n" number of trees**.

Random forest prediction pseudo code

1. Predict and store the outcome using the rules of each randomly created decision tree.
2. Count on the votes for each predicted target.
3. The last prediction is selected by taking the high voted predicted outcome.

With two trees, you can see how a random forest would look like

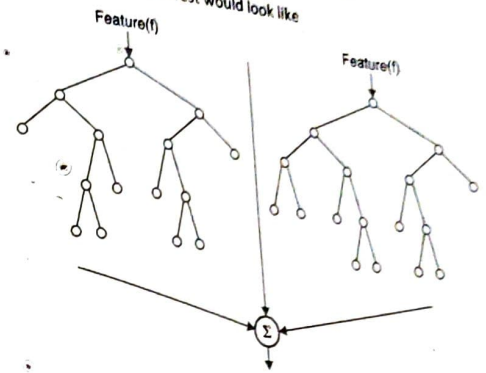


Fig. 5.1.6 : Random forest with 2 trees

Advantages

1. It can be used for both regression and classification tasks.
2. It is a very handy and easy to use algorithm as it's default hyper parameters often produce a good prediction result.
3. The classifier won't overfit the model if there are enough trees in the forest.

Disadvantages

1. Many trees are generated which makes algorithm slow and not suited for real time prediction.
2. It's not a descriptive tool, but only predictive model.
3. If dataset is noisy , then random forest may overfit.
4. For data, including categorical variables with different number of levels, random forests are biased in favour of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

5.1.4(B) Ada Boost

- It combines weak classifier algorithm to form strong classifier, then by selecting, training set at every iteration multiple classifiers are combined which gives good accuracy score.
- Correct amount of weight can be assigned in final voting, which improves accuracy.

Algorithm

Step 1 : Choose the training set and train the algorithm.

Step 2 : Retrains the algorithm iteratively by selecting another set of training data based on the accuracy of previous training.

Step 3 : The weight-age depends on the accuracy achieved for each trained classifier.

Step 4 : It assigns weight to each training item.

Step 5 : Higher weights are assigned to misclassified item so that those items can appear in the training subset of next classifier with higher probability.

Step 6 : After training weights is assigned to each classifier also based on accuracy.

Step 7 : Higher weights are assigned to more accurate classifier to get more impact on the final outcome.

Step 8 : The mathematical formula for Adaboost is given below.

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Where, T : Number of classifiers

$h_t(x)$: Output of weak classifier t for input x

α_t : Weight assigned to the classifier.

α_t is calculated as follows:

$\alpha_t = 0.5 * ln\ (\ (1 - E)\ /\ E\ )$ where E is error rate.

Step 9 : Initially, all the input training examples has equal weightage. The mathematical formula to update the weight of each training example is given below.

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where, D_t : Weight at previous level.

Z_t : Sum of all weights used to normalize the weights.

y_i is y par of training example (x_i, y_i) y coordinate for simplicity.

5.1.4(C) Gradient Tree Boosting

Gradient tree boosting is a technique that allows you to build a tree ensemble step by step with the goal of minimizing a target loss function. Gradient boosting has main three components:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

- Mean squared error (MSE) as loss defined as:

$$\text{Loss} = \text{MSE} = \Sigma (y_i - y_i^P)^2$$

Where, y_i = i^{th} target value, y_i^P = i^{th} prediction, $L(y_i, y_i^P)$ is loss function

- MSE should be minimum for our predictions.
- Use a gradient descent and update our predictions based on a learning rate, we can find the values where MSE is minimized.

$$y_i^P = y_i^P + \alpha * \delta \Sigma (y_i - y_i^P)^2 / \delta y_i^P$$

Which becomes, $y_i^P = y_i^P - \alpha * 2 * \Sigma (y_i - y_i^P)$

Where, α is learning rate and $\Sigma (y_i - y_i^P)$ is the sum of residuals.

Steps of Gradient Boost algorithm

Step 1 : Assume mean is the prediction of all variables.

Step 2 : For each observation calculate errors using mean (latest prediction).

Step 3 : Find the variable that can split the errors perfectly and find the value for the split. This is assumed to be the latest prediction.

Step 4 : Calculate errors of each observation from the mean of both the sides of the split (latest prediction).

Step 5 : Repeat the step 3 and 4 till the objective function maximizes/minimizes.

Step 6 : Take a weighted mean of all the classifiers to come up with the final model.

5.1.4(D) Voting Classifier

Voting classifier has two strategies

1. **Hard Voting :** Final class label is predicted based on the most frequent class label of classification models.
2. **Soft Voting :** Final class label is predicted by averaging the class probabilities.

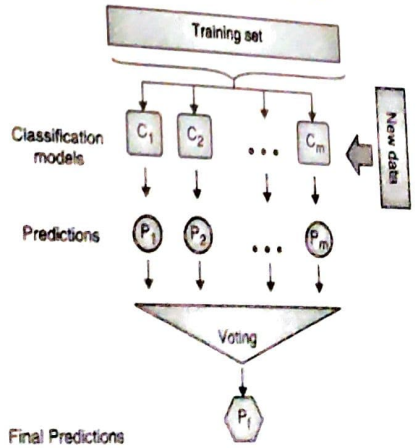


Fig. 5.1.7 : Voting classifier block diagram

Hard Voting or Majority Voting

Predict the class label Y via majority voting of each classifier C_j :

$$Y = \text{mode}\ \{C_1(x), C_2(x), \ldots, C_m(x)\}$$

Assume 3 classifiers and predicted class of each as follows :

classifier 1 > class 0

classifier 2 > class 0

classifier 3 > class 1

$$y = \text{mode}\ \{0,0,1\} = 0$$

So final predicted class for the sample is "0"

Weighted Majority Vote

Weighted majority vote can be calculated by associating a weight w_j with classifier C_j :

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j X_A (C_j(x) = i),$$

where X_A is the characteristic

A is the set of unique class labels.

Assume 3 classifiers and predicted class of each as follow :

classifier 1 > class 0

classifier 2 > class 0

classifier 3 > class 1

Assign the weights as {0.2, 0.2, 0.6} respectively for each class.

Prediction for Y is

$$\arg\max_i [0.2 \times i_0 + 0.2 \times i_0 + 0.6 \times i_1] = 1$$

Soft Voting

Class label is predicted based on the predicted probabilities p for the classifier. This works only if classifiers are well-calibrated.

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_i p_{ij},$$

Where, w_j is the weight that can be assigned to the j^{th} classifier.

Assume the same example with binary classification having class labels {0 , 1} our ensemble could make the following prediction :

C1(x) → [0.9,0.1]

C2(x) → [0.8,0.2]

C3(x) → [0.4,0.6]

Using uniform weights, we compute the average probabilities :

$$p(i_0 \mid x) = \frac{0.9 + 0.8 + 0.4}{3} = 0.7$$

$$p(i_1 \mid x) = \frac{0.1 + 0.2 + 0.6}{3} = 0.3$$

$$\hat{y} = \arg\max_i [p(i_0 \mid x), p(i_1 \mid x)] = 0$$

However, assigning the weights {0.1, 0.1, 0.8} would yield a prediction $\hat{y} = 1$:

$$p(i_0 \mid x) = 0.1 \times 0.9 + 0.1 \times 0.8 + 0.8 \times 0.4 = 0.49$$

$$p(i_1 \mid x) = 0.1 \times 0.1 + 0.2 \times 0.1 + 0.8 \times 0.6 = 0.51$$

$$\hat{y} = \arg\max_i [p(i_0 \mid x), p(i_1 \mid x)] = 1$$

5.2 Clustering Fundamentals

5.2.1 Basics of Clustering

- Clustering is an unsupervised learning problem.
- No classes or groups are known earlier.
- Data is divided into different groups, called as clusters.
- Clusters should be made based on data objects having high inter similarity and low intra similarity.
- In **hard clustering** techniques, each element must belong to a single cluster and in **soft clustering** or **fuzzy clustering**, is based on a membership score that defines how much the elements are "compatible" with each cluster.
- Graphical representation can be as shown in Fig. 5.2.1.

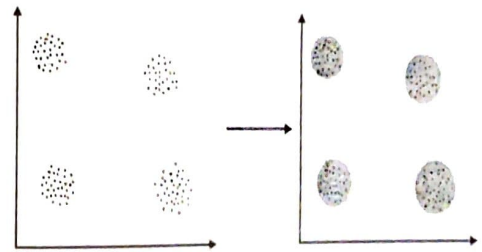


Fig. 5.2.1 : Clustering Graphical example

- From Fig. 5.2.1, we can observe there are 4 clusters based on their geometrical distance grouped together is called distance based clustering.
- Conceptual clustering is based on common or descriptive concepts of data objects.

Applications

Clustering algorithms can be applied in many disciplines :

- **Marketing :** Clustering can be used for targeted marketing. For e.g. Given a customer database, containing properties and past buying records. Similar groups of customers can be identified and grouped into one cluster.
- **Biology :** Clustering can also be employed in classifying plants and animals into different categories based on their characteristics.
- **Libraries :** books can be clusters based on their details.
- **Insurance :** Different groups of policy holders can be placed with clustering. For e.g. identify policy fraud or customers with high claim.
- **City-planning :** Groups of houses based on location and type.
- **Earthquake studies :** Clustering can also be used to identify dangerous zones based on earthquake epicentre.
- **WWW :** Clustering can be used to find groups of similar access patterns using weblog data. It can also be used for classification of documents.

5.2.2 K-means

- In 1967, J. MacQueen and then in 1975 J. A. Hartigan and M. A. Wong developed K-means clustering algorithm.
- In k-means, k is the number of clusters given by user and objects are classified into k clusters based on their attributes.
- K-means is one of the simplest unsupervised learning algorithms.
- Define K centroids for K clusters which are generally far away from each other.
- Group the objects into clusters based on the distance with respect to centroid
- After this first step, again calculate the new centroid for each cluster based on the elements of that cluster.
- Follow the same method and group the elements based on new centroid.
- At every step, the centroid changes and elements move from one cluster to another.
- Do the same process till no element is moving from one cluster to another i.e. till two consecutive steps with the same centroid and the same elements are obtained.
- Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function is given as follows,

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \| x_i^{(j)} - c_j \|^2$$

Where, $x_i^{(j)}$ = A data point

c_j = The cluster centre

n = Number of data points

k = Number of clusters

$\| x_i^{(j)} - c_j \|^2$ = Distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j

5.2.2(A) K-means Algorithm

k : Number of clusters

n : Sample feature vectors $x_1, x_2, ..., x_n$

m_i : The mean of the vectors in cluster i

Assume k < n.

Make initial guesses for the means $m_1, m_2, ..., m_k$.

Until there are no changes in any mean.

Use the estimated means to classify the samples into clusters.

for i = 1 to k

Replace m_i with the mean of all of the samples for cluster i

end_for

end_until

Following the three steps are repeated until convergence:

Iterate till no object moves to a different group :

Step 1 : Find the centroid coordinate.

Step 2 : Find the distance of each object to the centroids.

Step 3 : Based on minimum distance group the objects.

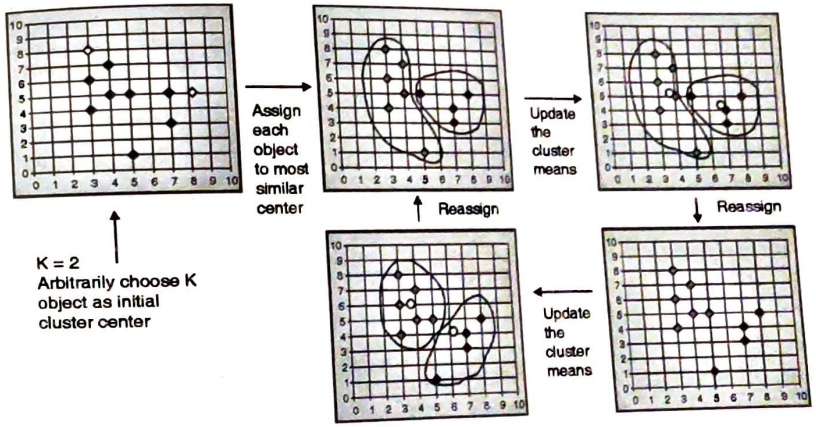


Fig. 5.2.2 : K-means graphical example

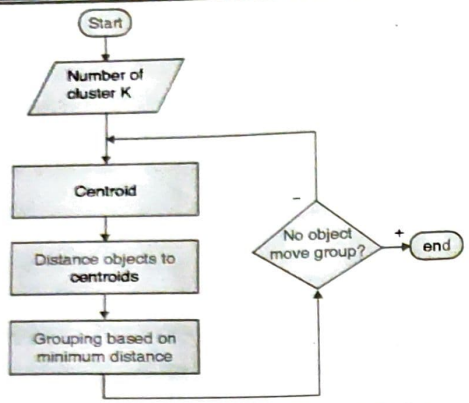


Fig. 5.2.3 : Basic steps for K-means clustering

Given a cluster $K_i = \{t_{i1}, t_{i2},...,t_{im}\}$, the cluster mean is $m_i = (1/m)(t_{i1} + ... + t_{im})$

5.2.2(B) Clustering with Scikit-learn

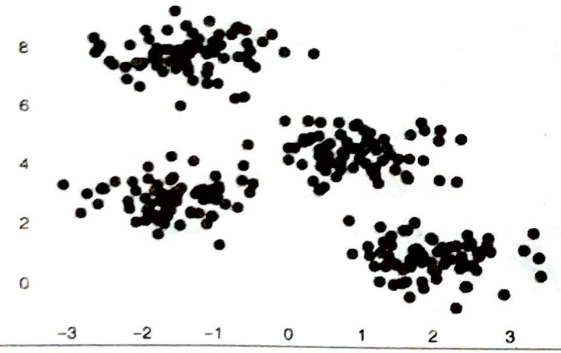
Let's consider a simple example with a dummy dataset. An example taken from https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

We begin with the standard imports :

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()# for plot styling
import numpy as np
```

First, let's generate a two-dimensional dataset containing four distinct blobs. To emphasize that this is an unsupervised algorithm, we will leave the labels out of the visualization.

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

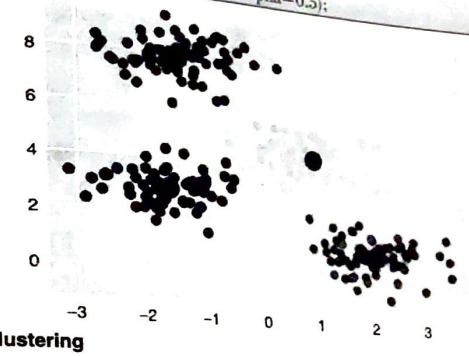



```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
print(kmeans.cluster_centers_)
```

```
[[-1.37324398  7.75368871]
 [-1.58438467  2.83081263]
 [ 1.98258281  0.86771314]
 [ 0.94973532  4.41906906]]
```

Let's visualize the results by plotting the data colored by these labels

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



Strength of K-means clustering

Relatively efficient : O(ikn),

Where, n is number of objects,
k is number of clusters,
i is number of iterations.

Normally, k, i<<n.

- K-means often terminates at a **local optimum**.
- Techniques like **deterministic annealing** and genetic algorithms are used to get the global optimum solution.

Weakness of K-means clustering

- Applicable only when **mean** is defined.
- Need to specify *k*, the number of clusters, in advance.
- Unable to handle noisy data and **outliers** (outlier: objects with extremely large values)
- Not suitable to discover clusters with **non-convex shapes.**

5.2.2(C) Finding Optimal Number of Clusters

Clustering techniques like K-Means clustering requires clustering parameter, the number of clusters K. This parameter plays an important role in the analysis, hence its important to decide the optimal value of K. If the number of clusters are less, then large element grouping can be formed with heterogeneous elements. With the more number of clusters, it will be difficult to get the dissimilarities among clusters.

Finding the right number of clusters is dependent on many factors like a) distribution shape b) scale in the data set. Following are the methods useful to find an optimal value of k

1. **Cross validation**
   - Divides the data into X parts, then trains the model on X – 1 parts and validates on remaining part.
   - Model is validated by checking the value of the sum of squared distance to the centroid. The final value is calculated by average over X clusters.
2. **Elbow method**
   - Value of K is calculated by the percentage of variance in each cluster.
   - The value of K selected is such that percentage of variance is maximum after generating a plot by the elbow method.
3. **Silhouette method**
   - This method is used to find the separation distance between resulting clusters.
   - The plot displays a measure of how close each point in a cluster is close to points in neighboring clusters.
   - This provides a way to find the number of clusters.
4. **X means clustering**
   - This technique is a variation of k means clustering.
   - It starts with number of clusters =1 and continues to divide the set of observations into clusters until best split is obtained or the stopping criterion is obtained.
   - For best split Bayesian criteria is used.

More methods to determine the number of clusters are

(A) Optimizing the inertia (B) Silhouette score

(C) Calinski-Harabasz index (D) Cluster instability

(A) Optimizing the inertia

- The assumption that an appropriate number of clusters must produce a small inertia.
- When number of clusters is same as number of data elements, this value reaches its minimum (0.0).
- So don't look for the minimum, but for a value which is a trade-off between the inertia and the number of clusters.
- This method calculates the inertias for different number of clusters.

(B) Silhouette score

- **Silhouette score** is a way to measure how the elements within a cluster are close to the points in its neighbouring clusters.
- It is based on the principal of "maximum internal cohesion and maximum cluster separation" means how similarly an object is in its own cluster (cohesion) compared to other clusters (separation).
- It finds the optimal value of k (number of clusters) during clustering.
- Define a distance metric and compute the average intra-cluster distance for each element :

$$a(\bar{x}_i) = E_{\bar{x}_j \in C}[d(\bar{x}_i, \bar{x}_j)] \; \forall \; \bar{x}_i \in C$$

- Calculate the average nearest-cluster distance which is the lowest inter-cluster distance.

$$b(\bar{x}_i) = E_{\bar{x}_j \in D}[d(\bar{x}_i, \bar{x}_j)] \; \forall \; \bar{x}_i \in C \text{ where } D = \arg\min\{d(C, D)\}$$

- The silhouette score for an element x_i is defined as :

$$S(\bar{x}_i) = \frac{b(\bar{x}_i) - a(\bar{x}_i)}{\max\{a(\bar{x}_i), b(\bar{x}_i)\}}$$

- This value of silhouette score is bounded between – 1 and 1.
- Score closer to 1 means assigned to the cluster correctly and score closer to – 1 is assigned to a wrong cluster. A score close to 0 means the point lies between almost at the boundary of both the clusters.

scikit-learn allows computing the average silhouette score to have an immediate overview for different numbers of clusters : The program is taken from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

```
from __future__ import print_function

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

print(__doc__)

# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close together.
X, y =make_blobs(n_samples=500, n_features=2, centers=4, cluster_std=1, center_box=(-10.0, 10.0),
shuffle=True, random_state=1)  # For reproducibility

range_n_clusters = [2, 3, 4, 5, 6]
```

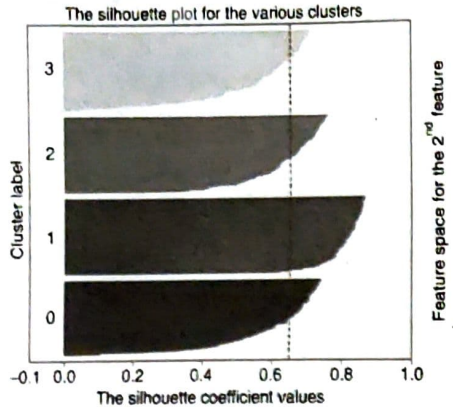
```
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) =plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot

    # The silhouette coefficient can range from – 1, 1 but in this example all lie within [– 0.1, 1]
    ax1.set_xlim([–0.1, 1])

    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.

    ax1.set_ylim([0, len(X) + (n_clusters +1) *10])

    # Initialize the clusterer with n_clusters value and a random generator seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
cluster_labels = clusterer.fit_predict(X)

# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the formed clusters
silhouette_avg =silhouette_score(X, cluster_labels)
print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values =silhouette_samples(X, cluster_labels)

y_lower =10
for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color,
        edgecolor=color, alpha=0.7)
```

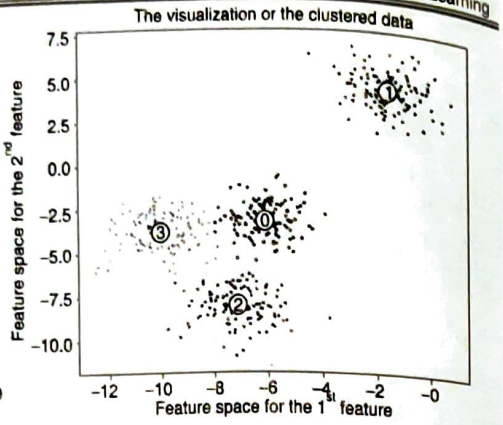


```
    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower +0.5* size_cluster_i, str(i))
    # Compute the new y_lower for next plot
    y_lower = y_upper +10# 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7, c=colors, edgecolor='k')

# Labeling the clusters
centers = clusterer.cluster_centers_

# Draw white circles at cluster centers
ax2.scatter(centers[:, 0], centers[:, 1], marker='o', c="white", alpha=1, s=200, edgecolor='k')
for i, c in enumerate(centers):
    ax2.scatter(c[0], c[1], marker='$%d$'% i, alpha=1, s=50, edgecolor='k')
ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")
plt.suptitle(("Silhouette analysis for KMeans clustering on sample data " "with n_clusters = %d"% n_clusters),
        fontsize=14, fontweight='bold')
plt.show()
```

Output

```
For n_clusters = 2 The average silhouette_score is : 0.7049787496083261
For n_clusters = 3 The average silhouette_score is : 0.5882004012129721
For n_clusters = 4 The average silhouette_score is : 0.6505186632729437
For n_clusters = 5 The average silhouette_score is : 0.56376469026194
For n_clusters = 6 The average silhouette_score is : 0.4504666294372765
```





Similar graphs are generated for 2, 3, 5, 6 number of clusters.

(C) Calinski-Harabasz index

- It is a concept of dense and well-separated clusters.
- It is required to define the inter cluster dispersion initially.
- The inter-cluster dispersion (BCD) is defined as :

$$BCD(k) = Tr(B_k) \qquad \text{where, } B_k = \sum_t n_t (\mu - \mu_j)^T (\mu - \mu_j)$$

- Where k is the cluster with their relative centroids μ_i and the global centroid μ , n_t is the number of elements belonging to the cluster k.
- The intra-cluster dispersion (WCD) is defined as :

$$WCD(k) = Tr(X_k) \qquad \text{where, } X_k = \sum_t \sum_{x \in C_k} (x - \mu_t)^T (x - \mu_t)$$

- The Calinski-Harabasz index is defined as the ratio between BCD(k) and WCD(k) :

$$CH(k) = \frac{N-k}{k-1} \cdot \frac{BCD(k)}{WCD(k)}$$

- We have to find the number of clusters which gives a maximum index Calinski-Harabasz index
- The code to find Calinski-Harabasz index is given below for iris dataset. Change the number of clusters in code and check the index.

```
# Refer : https://scikit-learn.org/stable/modules/clustering.html
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn import datasets
dataset = datasets.load_iris()
X = dataset.data
y = dataset.target
```



```
# the Calinski-Harabaz index is applied to the results of a cluster analysis
import numpy as np
from sklearn.cluster import KMeans
# Number of clusters = 3
kmeans_model = KMeans(n_clusters=3, random_state=1).fit(X)
labels = kmeans_model.labels_metrics.calinski_harabaz_score(X, labels)
```

Output

```
560.3999242466402
```

```
# Number of clusters = 5
kmeans_model = KMeans(n_clusters=5, random_state=1).fit(X)
labels = kmeans_model.labels_
metrics.calinski_harabaz_score(X, labels)
```

Output

```
494.0943819140987
```

(D) Cluster instability

- It is based on the concept of cluster instability defined in Von Luxburg U.
- If we have a dataset X, we can define a set of m perturbed (or noisy) versions :

$$X_n = \{X_n^0, X_n^1, \ldots, X_n^m\}$$

- Considering a distance metric $d(C(X_1), C(X_2))$ between two clusterings with the same number (k) of clusters, the instability is defined as the average distance between couples of clusterings of noisy versions :

$$I(C) = E_{x_n^{i,j} \in X_n} [d(C(X_n^i), C(X_n^j))]$$

- We need to find the value of k that minimizes I(C) (and therefore maximizes the stability).

5.2.3 DBSCAN

DBSCAN : Density Based Spatial Clustering of Applications with Noise

The algorithm DBSCAN, based on the formal notion of density-reachability for *k*-dimensional points, is designed to discover clusters of arbitrary shape. The runtime of the algorithm is of the order O(n log n) if region queries are efficiently supported by spatial index structures, i.e. at least in moderately dimensional spaces.

Explanation of DBSCAN Steps

- Epsilon (Eps) and Minimum points (MinPts) are the two parameters needed by DBSCAN. An unvisited point is chosen as the starting point. Eps between the starting point and its neighbors is calculated and the points within it are considered.
- If the number of data points in the neighbourhood is greater than or equal to MinPts then a cluster is formed. The starting point chosen is marked as visited.
- The above steps are then repeated for all the remaining neighbours.

- If the data points found in the neighbourhood are less than MinPts then they are marked as noise.
- If all the points within reach in a cluster are visited then the algorithm proceeds by choosing other remaining invested points in the dataset.

Basic concept

For any cluster, we have :

- A central point (p) i.e. core point.
- A distance from the core point (Eps).
- Minimum number of points within the specified distance (MinPts).

Major features

(i) Discover clusters of arbitrary shape.

(ii) Handles noise.

(iii) One scan.

(iv) Need density parameters as termination condition.

DBSCAN method

- Clusters of arbitrary shape and size are grown which are dense.
- The algorithm is as follows :
  (1) Core objects and density reachable objects are found out, merge these density reachable core objects and their clusters are discovered.
  (2) When no new points can be added to any of the clusters the execution may be stopped.
- Clusters are dense regions of objects separated by regions of low density (noise).
- Outliers will not affect the creation of cluster.

Input

(1) MinPts : Minimum number of points in any cluster.

(2) ∈ : For each point in the cluster there must be another point in its less than this distance away.

∈-neighborhood : Points within ∈ distance of a point.

N∈(p) : {q belongs to D | dist(p,q) <= ∈}

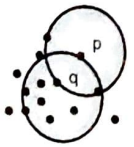
Core point : ∈ - neighborhood dense enough (MinPts)

Directly density-reachable : A point p is directly density-reachable from a point q if the distance is small (∈) and q is a core point.

(1) p belongs to N∈(q)

(2) core point condition :

$|N_{\in}(q)| > = MinPts$



MinPts = 5

Eps = 1cm

Issues

- One of the limitations is that, the user has to set the MinPts and Eps threshold. This needs a good knowledge of the dataset. Sometimes in high dimensional data set, it is difficult to decide.
- Some of the dataset distribution may be globally inconsistent. For example some of the area in the dataset may be too dense compared to the other areas, some of the sections may not have clusters or noise may be present.
- The process is extremely sensitive to noise which leads to very different clusters.

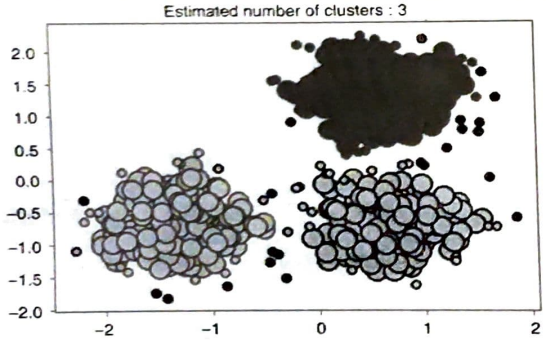
Code for DBSCAN (https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html)

```
print(__doc__)
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
# ##############################################################################
centers=[[1,1],[-1,-1],[1,-1]]
X,labels_true=make_blobs(n_samples=750,centers=centers,cluster_std=0.4, random_state=0)

X=StandardScaler().fit_transform(X)

# ##############################################################################
db=DBSCAN(eps=0.3,min_samples=10).fit(X)
core_samples_mask=np.zeros_like(db.labels_,dtype=bool)
core_samples_mask[db.core_sample_indices_]=True
labels=db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_=len(set(labels)) - (1if-1inlabelselse0)
n_noise_=list(labels).count( - 1)

print('Estimated number of clusters: %d' %n_clusters_)
print('Estimated number of noise points: %d' %n_noise_)
print("Homogeneity: %0.3f" %metrics.homogeneity_score(labels_true,labels))
print("Completeness: %0.3f" %metrics.completeness_score(labels_true,labels))
print("V-measure: %0.3f"%metrics.v_measure_score(labels_true,labels))
print("Adjusted Rand Index: %0.3f" %metrics.adjusted_rand_score(labels_true,labels))
print("Adjusted Mutual Information: %0.3f" %metrics.adjusted_mutual_info_score(labels_true,labels))
print("Silhouette Coefficient: %0.3f" %metrics.silhouette_score(X,labels))
# ############################################################################## Plot result
import matplotlib.pyplot as plt
# Black removed and is used for noise instead.
```

```
unique_labels=set(labels)
colors=[plt.cm.Spectral(each)
                for each in np.linspace(0,1,len(unique_labels))]
for k,col in zip(unique_labels,colors):
    if k==-1:
      # Black used for noise.
      col=[0,0,0,1]

    class_member_mask=(labels==k)

    xy=X[class_member_mask&core_samples_mask]
    plt.plot(xy[:,0],xy[:,1],'o',markerfacecolor=tuple(col),markeredgecolor='k',markersize=14)
    xy=X[class_member_mask&~core_samples_mask]
    plt.plot(xy[:,0],xy[:,1],'o',markerfacecolor=tuple(col),markeredgecolor='k',markersize=6)
plt.title('Estimated number of clusters: %d' %n_clusters_)
plt.show()
```

Output

```
Automatically created module for IPython interactive environment
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.883
Silhouette Coefficient: 0.626
```



5.2.4 Spectral Clustering

The goal of spectral clustering is to cluster data that is connected but not necessarily compact or clustered within convex boundaries.

Three main steps of spectral clustering :

1. Create a similarity graph between our N objects to cluster.
2. Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object.
3. Run k-means on these features to separate objects into k classes.
   - Compactness, e.g. k-means, mixture models
   - Connectivity, e.g. spectral clustering

Compactness Connectivity

Charles H Martin (https://calculatedcontent.com/2012/10/09/spectral-clustering/) has described the algorithm as follows :

1. project your data into R^n
2. define an Affinity matrix A, using a Gaussian Kernel K or say just an Adjacency matrix (i.e. $A_{ij} = \delta_{i,j}$).
3. construct the Graph Laplacian from A (i.e. decide on a normalization).
4. solve an Eigen value problem , such as $L_v = X_v$ (or a Generalized Eigen value problem $L_v = \lambda D_v$).
5. select k eigenvectors { v_i, i = 1, k} corresponding to the k lowest (or highest) eigenvalues { λ_i, i = 1, k}, to define a k-dimensional subspace P^t LP.
6. form clusters in this subspace using, say, k-means.

5.3 Evaluation Methods Based on Ground Truth- Homogeneity, Completeness, Adjusted Rand Index

Before the prediction of any new sample, evaluation of model is necessary. The model can be evaluated with Homogeneity, Completeness, Adjusted Rand Index.

1. Homogeneity (h)

The homogeneity score is used to determine the optimal number of clusters. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

$$h = 1 - \frac{H(C \mid K)}{H(C)}$$

Where,

C : Class set , K : Cluster set

H(C|K) : Measure of uncertainty after clustering in determining the right class. it's necessary to normalize this value considering the initial entropy of the class set H(C).

2. Completeness (c)

Each sample of the same class must belong to the same cluster. Completeness can be measured using conditional entropy H(K|C). It's necessary to normalize this using entropy H(K). The value should be high to evaluate the number of clusters selected. If the value is high, then the majority of samples of same class assigned to the same cluster.

$$c = 1 - \frac{H(K|C)}{H(K)}$$

3. Adjusted rand index

The adjusted rand index measures the similarity between the original class partitioning and the clustering.

a : The number of pairs of elements belonging to the same partition in the class set C and to the same partition in the clustering set K number of pairs of elements belonging to different partitions in the class set C and to different partitions in the clustering set K.

If our total number of samples in the dataset are n, the rand index is defined as :

$$R = \frac{a+n}{\binom{n}{2}}$$

The Corrected for Chance version is the adjusted rand index, defined as follows :

$$AR = \frac{R - E[R]}{\max[R] - E[R]}$$

As the adjusted rand score is bounded between – 1.0 and 1.0, with negative values representing a bad situation.

The code for Evaluation methods :

https://scikit-learn.org/stable/auto_examples/cluster/plot_affinity_propagation.html

#sphx-glr-auto-examples-cluster-plot-affinity-propagation-py

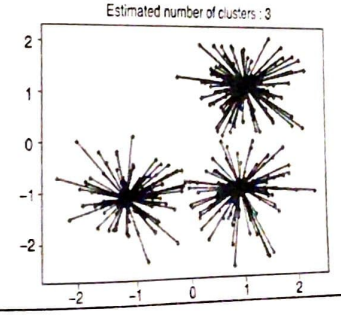
```
print(__doc__)
from sklearn.cluster import AffinityPropagation
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
# ############################################## Generate sample data
centers=[[1,1],[-1,-1],[1,-1]]
X,labels_true=make_blobs(n_samples=300,centers=centers,cluster_std=0.5, random_state=0)
# ############################################## Compute Affinity Propagation
af=AffinityPropagation(preference=-50).fit(X)
cluster_centers_indices=af.cluster_centers_indices_
labels=af.labels_
n_clusters_=len(cluster_centers_indices)
print('Estimated number of clusters: %d'%n_clusters_)
print("Homogeneity: %0.3f"%metrics.homogeneity_score(labels_true,labels))
print("Completeness: %0.3f"%metrics.completeness_score(labels_true,labels))
print("V-measure: %0.3f"%metrics.v_measure_score(labels_true,labels))
```



```
print("Adjusted Rand Index: %0.3f"%metrics.adjusted_rand_score(labels_true,labels))
print("Adjusted Mutual Information: %0.3f" %metrics.adjusted_mutual_info_score(labels_true,labels))
print("Silhouette Coefficient: %0.3f" %metrics.silhouette_score(X,labels,metric = 'sqeuclidean'))
# ############################################################################# Plot result
import matplotlib.pyplot as plt
from itertools import cycle
plt.close('all')
plt.figure(1)
plt.clf()
colors=cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
for k,col in zip(range(n_clusters_),colors):
    class_members=labels==k
    cluster_center=X[cluster_centers_indices[k]]
    plt.plot(X[class_members,0],X[class_members,1],col+'.')
    plt.plot(cluster_center[0],cluster_center[1],'o',markerfacecolor=col,markeredgecolor='k',markersize=14)
    for x in X[class_members]:
        plt.plot([cluster_center[0],x[0]],[cluster_center[1],x[1]],col)
plt.title('Estimated number of clusters: %d'%n_clusters_)
plt.show()
```

Output

```
Automatically created module for IPython interactive environment
Estimated number of clusters: 3
Homogeneity: 0.872
Completeness: 0.872
V-measure: 0.872
Adjusted Rand Index: 0.912
Adjusted Mutual Information: 0.871
Silhouette Coefficient: 0.753
```



5.4 Introduction to Meta Classifier

- Meta classifier is a classifier, which is usually a proxy to the main classifier, used to provide additional data pre-processing.
- Meta classifier doesn't implement a classification algorithm on its own, but uses another classifier to do the actual work. In addition, the meta-classifier adds another processing step that is performed before the actual base-classifier sees the data.
- Use meta classifier before the classification on both training and testing data. The Meta classifier technique is used to implement arbitrary pre-processing within a cross-validation analysis.

5.4.1 Concepts of Weak/Lazy and Eager Learner

Weak /Lazy Learner:

- Simply store training data (with labels) and wait until a new instance arrives that needs to be classified.
- Generalizing beyond these data is postponed until an explicit request is made.
- Can construct a different approximation to the target function for each encountered query instance.
- No model is constructed.
- Also known as instance-based learners, because they store the training set.
- **Example :** K-NN classifier, case based reasoning.

Eager learner:

- Construct a classification model (based on training data).
- It constructs general, explicit description of the target function based on the provided training samples.
- Use the same approximation to the target function, which must be learned based on training examples and before input queries are observed.
- Learned models are ready and eager to classify as previously unseen instances.
- **Example :** Decision trees, Support vector machines, Bayesian classification, Rule based classification and classification based on association rule mining, classification by back propagation .

Lazy Learner Classification Methods

K-Nearest-Neighbor Classifiers

- KNN classifiers learn by comparing a given test tuple with training tuples that are similar to it.
- KNN classifier searches the pattern space for the k-training tuples that are closest to the unknown tuple.
- These training tuples are k-nearest neighbours of the unknown tuple

Case-Based Reasoning

- This method first checks for an identical training case, if one exists then it is returned.
- If no identical case exists than the method will search for a similar cases having similar components to those of a new case.
- If cases are represented as graphs, then the process involves searching for sub graphs within the new case.

Eager Learner Classification Methods

Classification by Decision Tree Induction

- A decision tree is a structure in which internal node denotes a test attribute, each branch represents an outcome and each leaf node holds a class label.
- The topmost node in the tree is known as root node.
- ID3, C4.5 and CART are top-down recursive, divide and conquer approaches.
- It starts with the training set of tuples and their class labels, the training set is recursively partitioned into smaller subsets as the tree is been built.

Bayesian Classification

- This classification method is based on statistical concept.
- For prediction a class membership probability is calculated to find whether a given tuple belongs to a particular class.
- They exhibit high accuracy when applied to large datasets.
- The classifier uses an assumption of class conditional independence, which means an effect of an attribute value on a given class is independent of the value of other attributes.

Review Questions

Q.1 Explain different impurity measures.

Q.2 What is Ensemble Learning?

Q.3 Explain following algorithms in detail

(a) Random Forest (b) Ada Boost

(c) K-Means (d) DBSCAN

Q.4 What are different methods to determine number of clusters? Explain.

□□□